Reference Manual for the NUFT Flow and Transport Code, Version 1.0

John J. Nitao
Earth Sciences Department
Lawrence Livermore National Laboratory

date: March 20, 1996

time: 12:00 AM

file: ref.tex

Contents

1. Introduction

NUFT (Nonisothermal Unsaturated-Saturated Flow and Transport model) is a suite of multi-phase, multicomponent models for numerical solution of non-isothermal flow and transport in porous media with application to subsurface contaminant transport problems. These distinct models are imbedded in a single code in order to utilize a common set of utility routines and input file format.

Currently, the code runs on the Unix and DOS operating systems. Versions have been successfully compiled and tested for IBM-PC compatibles, Cray Unicos, and the following workstations: Sun, Hewlett-Packward, IBM Risc/6000, Silicon Graphics, DEC Alpha. Each set of related models is called a module and has its own user's manual which documents any particular features and input data specific to that module. The NUFT Reference Manual [Nitao, 1993] documents the general numerical algorithms used and gives the documentation of the input to the model common to all or most modules including options not described in the user's manual for each module.

Available modules are:

- `UCSAT` – unconfined and confined saturated flow model
- `US1P` – single phase unsaturated flow (Richard's equation)
- `US1C` – single component contaminant transport
- `USNT` – NP-phase, NC-component with thermal option.

An integrated finite-difference spatial discretization is used to solve the balance equations. The resulting non-linear equation is solved at each time by the Newton-Raphson method. Options for solution of the linear equations at each iteration are direct banded solution and preconditioned conjugate gradient method with various preconditioning schemes.

The model can solve one- , two- , or three-dimensional problems. Future plans include incorporation of capillary hysteresis, non-orthogonal mesh discretization, finite elements, non-linear solid sorption isotherms, and chemical reactions.

The first stage of code verification with one-dimensional problems has been completed (Lee et al., [1993]) and further verification efforts are planned.

The distinct models in the code utilize a common set of utility routines and input file format. The various models are essentially isolated from each other and hence future models can be added without affecting existing models. This also allows for easy maintenance and future enhancements. Global variables in the code are virtually non-existent. The code is written principally in the C language. Input data is in the form of that used by the lisp language. An internal lisp interpreter for the Scheme dialect of lisp is part of the simulator whose purpose is to read the input data file and the internal data files containing default input data values. It also performs data checking.

This manual is self-contained and describes a minimal set of the most commonly used input parameters necessary for the use of this module. The NUFT reference manual (Nitao, [1995]) contains generic input options common to all of most of the modules. It also contains further input options not given in this manual.

Each module has its own user's manual documenting the data input specific to each module. Non-module-specific input are also given in the user's manual but with only the most commonly used input options covered. This reference manual describes the options for the non-module-specific data input in more detail and, also, describes the general numerical algorithms used.

acknowledgements

The author wishes to thank the management of the Environmental Restoration Division at the Lawrence Livermore National Laboratory (LLNL) for supporting the documentation and verification of the NUFT code. The basic concepts of the code was developed under the funding of the LLNL Institutional Research and Development program.

2. The Syntax of the Input Data

This section describes the syntax of the input data. The input data file format is in free format, i.e. it does not matter what column the data is in, nor does it matter if the data is continued past the current line or lines.

Input consists of lists of *data blocks, or data units.* Each data unit starts with a left parenthesis and ends with a right parenthesis. A data unit is of the following general form:

(*<name>* *<data>* *<data>* ...)

where

<name> refers to the input "variable" that is being set or specified

<data> are items which are *real numbers, integer numbers, time real numbers, strings, pattern strings, words,* or other data units, or *list(s)* of data items.

The different data types are defined later in this section.

An alternate form for a data unit is

(\ *<name>* *<data>* *<data>* ... \ *<name>*)

An advantage of this form is that the model can more reliably tell the user the exact location of any unmatching parentheses.

Example:

```
(porosity 0.2)
(file-name  "input.data")
(par  0.1 0.3 0.6)
```

This example sets three different variables. It sets the variable `porosity` to the numeric value, `0.2`, the variable `file-name` to the string, `"input.data"`, and the variable `par` to a list of three numeric values, `0.1 0.3 0.6`.

Example:

```
(\rocktab  (silt  (porosity 0.3) (Kx 1.e-4)(Ky 1.e-4)(Kz 1.e-4))
          (sand  (porosity 0.2) (Kx 1.e-2)(Ky 0.0)(Kz 0.0))
          (clay  (porosity 0.4) (Kx 1.e-6)(Ky 0.0)(Kz 0.0))
 \rocktab)
```

This example shows how a data unit which sets the variable, `rocktab`, to a list of data units.

Comment Character: Semi-colons in the input file serve as comment characters. That is, all characters on a given line after a semicolon are ignored by the program. Using comments is a good way for the user to annotate his input file. Using two semicolons instead of a single one is a good way to make sure that comments standout.

Example:

```
(porosity 0.2)  ;; this is how we set the value of porosity to 0.2
```

Units: All quantities are in MKS units (i.e. meters, kilogram, seconds). Thus, *hydraulic conductivities* are in meters/second, and *head* is in meters (see Table). Unitless quantities such

Table 1 Table of Units used in Input to Models

length	*meters (m)*
mass	*kilograms (kg)*
time†	*seconds (s)*
temperature	*centigrade* ($^\circ C$)
area	m^2
volume	m^3
mass density	kg/m^3
molar density	$mole/m^3$
permeability	m^2
hydraulic conductivity	m/s
flow velocity	m/s
force	*Newton* (Nt=kg-m/s^2)
pressure	*Pascals* (Pa=Nt/m^2)
head	m
energy	*Joule* (J=Nt-m)
specific energy	J/kg
mass flux	kg/s
molar flux	$mole/s$
volumetric flux	m^3/s
energy flux	*Watts* (W=J/s)
thermal conductivity	W-$m/{}^\circ C$
dynamic viscosity	Nt-s/m^2=kg/m-s=10^3 *centipoise*
molecular diffusivity	m^2/s

†model can accept other time units using unit designators

as saturation, porosity, and concentrations are always fractional (i.e. between 0 to 1, inclusive), *not* percentages.

Legal Data Types: We now describe the following valid data types.

1. A *string* is any sequence of visible characters delimited by double quotes “"”, for example,

   ```
   "hello there"
   " run3-B (test#2) "
   ```

   Note that spaces and parentheses are allowed in a string.

2. An *integer number*, for example, `11`.

3. A *real number* that is fixed or floating point. For example `1.23, -4.5e7, or 900.2E7`. Note that **D or d exponents in the manner of FORTRAN are not allowed !**

4. A *time number* which is a real number but with with the following unit designators as the last letter in order to denote units of time. As you may have guessed, this type of number is used whenever we want to specify a time.

s	seconds
m	minutes
h	hours
d	days
M	months
y	years

   If no unit designator is present, then seconds is assumed.

   Examples:

20.0	20 seconds
23.1s	23.1 seconds
45e4M	45e4 months

   There must be no spaces between the number and the unit designator.

5. A *word* is a sequence of non-blank visible characters. A word can be a variable or may be used in the same way as a string except that it can not have internal blanks. The model treats the words and strings as being distinct data types; the correct one as specified in the documentation is required.

6. A *pattern string* is a special type of a string with the two unix shell type “wild” characters

   `*` and `?`

   so that a pattern string can represent an entire class of strings that matches the string pattern. The character `*` in a pattern matches any sequence of characters. Hence, the pattern `"*"` matches all strings. The character `?` in a pattern matches any single character. Hence, the patter `"?"` matches all strings with exactly one character.
   Other Examples:

   1. The pattern `"ex*"` matches all strings that begin with the characters `"ex"`.
   2. The pattern `"ex*b2*z"` matches all strings that begin with `"ex"`, followed by any number (including zero) of strings which are then followed by the string `"b2"`, and which end with the string `"z"`.
   3. The pattern `"r2?xay"` matches all strings that begin with `"r2"` followed by a single character, and then followed by the characters `"xay"`.

Include statement:

The include statement is of the form

(include "<*file-name*>")

It is used to place the contents of the file with the name, "<*file-name*>", into the input file. The file must lie in the working directory under which NUFT is being run. It can only be used to replace a complete list, i.e. must be either a collection of data which is delimited by a closed sets of parentheses or a single data item such as a number or string. For example, suppose the file "data1.inc"contains

```
(field (format list) (range "*") (variables Sl)(file-ext ".Sl")
       (outtimes 0 70m 102m 222m 287m 342m 23h)
)
```

and the file "data2.inc"contains the single entry

```
200m
```

with the file "data3.inc"containing the string

```
".Sl"
```

Then, the following input data

```
(output
     (include " data1.inc")
     (forcetimes (outtimes (include " data2.inc" 201m))
)
```

will be interpreted by the model as begin equivalent to

```
(output
     (field (format list) (range "*") (variables Sl)(file-ext ".Sl")
            (outtimes 0 70m 102m 222m 287m 342m 23h)
     )
     (forcetimes (outtimes 200m 201m))
     (file-ext ".Sl")
)
```

The following is an example of an error. Suppose the file "file.inc"contains the following

```
(outtimes 0 70m 102m 222m
```

and the input file as the data item

```
(output
     (field (format list) (range "*") (variables Sl)(file-ext ".Sl")
```

```
                (include " file.inc")  287m 342m 23h)
        )
        (forcetimes (outtimes 200m 201m))
        (file-ext ".S1")
    )
```

This is an error because only complete lists or a single entry can be included (not to mention the fact that the parentheses will not match in the input file).

Include package statement:

The include package statement is of the form

`(include-pkg "`<*file-name*>`")`

This statment is identical to the `include` statement except that it includes a file from the subdirectory which contains the NUFT executable instead from the current working directory. The main purpose is to include a 'package' of pre-defined input parameters which comes with the NUFT software distribution.

Macro commands:

Macro commands start with the character '#'. There are three commands available: `#define`, `#ifdef`, and `#ifndef`. The following command defines a macro variable,
`(#define `<*variable*>`)`
Currently, a variable cannot be defined to be any particular value, but it is used in conjunction with the other macro commands. The statements within
`(#ifdef `<*variable*>` ....  )`
will be placed as part of the input stream if <*variable*> is defined by the `#define` command; whereas, the statements within
`(#ifndef `<*variable*>` ....  )` will not be placed as input if <*variable*> is not defined. The `#define` statements must be in the same parenthesis level as, for example, `bctab`, `genmsh`, etc. The `#ifdef` and `#ifndef` commands can be placed anywhere, except that the body of statements in the conditional commands must be complete lists, i.e. parentheses match inside the macro command. Currently, the macro commands only work inside an input set for a module or inside `common`.

3. How to Read the Input Documentation

The documentation of the input data to the code is written with special symbols which are not actually part of the data input but are used as a convenient short-hand to mean certain things.

The following is a list of special symbols that are used:

1. any word starting with the symbol < and ending with >
2. the symbols:

$$\| \qquad \ldots \qquad \{ \qquad \} \qquad [\qquad]$$

The meaning of these symbols is given as follows.

1. Any italized word starting with the character '<' and ending with the character '>' represents *data* as described in the previous section and will be called a *data token* or *token* for short.
2. [. . .] is an abbreviation that means that more data items follow but are not specified at this point; further explanation of the required missing data items will follow
3. [ ] means that data items inside [] are optional; for example, [(xyz *<real>*)]means that the input value of variable xyz is optional
4. || represents a logical "exclusive or" of two sets of data items; for example,

   (xyz *<real>*) || (abc *<real>*) means that the user *must* specify either the variable **xyz** or **abc** but not both

   [(xyz *<real>*) || (abc *<real>*)] means that the user has the *option* of either specifying **xyz** or **abc**
5. { } denotes a grouping of data items, usually used in conjunction with || ; for example:

   (xyz *<real>*) || { (abc *<real>*) (ijk *<integer>*) } means that the user must either specify **xyz** or, alternatively, the user must specify both **abc** *and* **ijk**

In the input documentation the following data tokens have special meanings:

<string>	a string
<integer>	an integer number
<real>	a real number
<t-real>	a time real number, the last character is alphabetic and denotes the units of time
<word>	a symbolic word
<pattern>	a pattern string

These data types are described in a previous section.

4. Basic Elements of the Input File

Before going further, the user should have read the previous sections explaining the abbreviations and special symbols used in this input documentation.

General Form of Input Data:

```
(<name-of-model>
     (title ...)       ;; run title
     (outputfile ...) ||
            { (output-prefix ...)(output-ext ...) }  ;; output file name
     (meshfile ...) ||  (genmsh ...)   ;; mesh specification
     (time ...)        ;; initial time
     (tstop ... )      ;; ending time
     (dt ...)          ;; initial time step
     (dtmax ...)       ;; maximum time step
     (stepmax ...)     ;; maximum no. of time steps
     (read-restart ...) ;; read from restart file
     (state ...)       ;; set initial conditions
     (rocktab ...)     ;; soil property type
     [ (grav-factor ...) ] ;; factor multiplying gravity vector

     [ (output...) ]    ;; output specification
     [ (srctab ...) ]   ;; source tables
     [ (bctab ...) ]    ;; boundary condition tables
)
```

Recall that the '...' denote subsequent data items which are to be explained later and that all of the input line past a semicolon is not read by the program but is for placing comments into the input file. The above data units do not have to occur in the above, or in any, particular order. The data entry <name-of-model> refers to the name of the model that is being used. For example 'us1p refers to the single-phase unsaturated flow model.

Note that the use of the square brackets around grav-factor, (output ...), (srctab ...), and (bctab ...) denote that these data units are optional. There are more optional data items which will be described later but the above ones are the most likely to be used. Initial conditions are set either using read-restart or state. One, but not both, of them must be present.

The above applies to NUFT modules that solve for flow and transport simultaneously. Some NUFT modules have the option of solving flow and transport sequentially. That is, the code first solves for the flow of phases and then the transport equation for the contaminant(s) is solved at each major time cycle in an alternating fashion. (The transport may take several time steps in a single major time cycle, if transport takes place at a much shorter time scale than flow.)

The form that is used when flow and transport is solved sequentially:

```
(common
     (title ...)
     (outputfile ...) ||  { (output-prefix ...)(output-ext ...) }
     (meshfile ...) ||  (genmsh ...)
     (time ...)
     (tstop ...)
)

(<name-of-flow-model>
     (dt ...)
```

```
        (dtmax ...)
        (stepmax ...)
        (state ...)
        (rocktab ...)

        [ (output...) ]
        [ (srctab ...) ]
        [ (bctab ...) ]
    )

    (<name-of-transport-model>
        (dt ...)
        (dtmax ...)
        (stepmax ...)
        (state ...)
        (rocktab ...)

        [ (output...) ]
        [ (srctab ...) ]
        [ (bctab ...) ]
    )
```

Note that the flow and transport models each have their own initial and maximum time steps as well as other data. Any input data that is common to both models are placed in the `(common` ... data unit. Any of the items in the common data unit can also appear in the data unit of the particular model but will be overridden by any specification that appears in the common data unit.

Include statement:

Input can be read from another file and placed into the input file using the `include` statement. The include statement is of the form

`(include "`<*file-name*>`")`

It is used to place the contents of the file with the name, "<*file-name*>", into the input file. It can only be used to replace a complete list, i.e. must be either a collection of data which is delimited by a closed sets of parentheses or a single data item such as a number or string. For example, suppose the file "data1.inc"contains

```
(field (format list) (range "*") (variables Sl)(file-ext ".Sl")
       (outtimes 0 70m 102m 222m 287m 342m 23h)
)
```

and the file "data2.inc"contains the single entry

```
200m
```

with the file "data3.inc"containing the string

```
".Sl"
```

Then, the following input data

```
(output
     (include " data1.inc")
     (forcetimes (outtimes (include " data2.inc" 201m))
)
```

will be interpreted by the model as begin equivalent to

```
(output
     (field (format list) (range "*") (variables Sl)(file-ext ".Sl")
            (outtimes 0 70m 102m 222m 287m 342m 23h)
     )
     (forcetimes (outtimes 200m 201m))
     (file-ext ".Sl")
)
```

The following is an example of an error. Suppose the file "file.inc"contains the following

```
(outtimes 0 70m 102m 222m
```

and the input file as the data item

```
(output
     (field (format list) (range "*") (variables Sl)(file-ext ".Sl")
            (include " file.inc")  287m 342m 23h)
     )
     (forcetimes (outtimes 200m 201m))
     (file-ext ".Sl")
)
```

This is an error because only complete lists or a single entry can be included (not to mention the fact that the parentheses will not match in the input file).

5. Input Data Documentation

The items in the input data file are classified in the following categories.

Mesh Generation Parameters

```
(genmsh ...)
(mesh-file ...)
```

Time Stepping and Numerical Solution Parameters

```
(time ...)
(tstop ...)
(dt ...)
(dtmax ...)
(stepmax ...)
```

Output Specification

```
(title ...)
(output ...)
```

Specification of Initial Conditions

```
(state ...)
(read-restart ...)
```

Rock Property Specification

```
(rocktab ...)
```

Source Term Specification

```
(srctab ...)
```

Boundary Condition Specification

```
(bctab ...)
```

Other options

```
(upstream-weighting ...)
(include ...)
```

NAME

genmsh – internally generate a rectangular or cylindrical grid system

SYNOPSIS

```
(genmsh
     (coord  <coord-type>)
     (down <num> <num> <num>)
     (dx <numx-0> <numx-1> ...)
     (dy <numy-0> <numy-1> ...)
     (dz <numz-0> <numz-1> ...)
     (mat
          (<el-name-prefix><mat-type>
               <i0><i1><j0><j1><k0> <k1>)
                    ...
          (<el-name-prefix><mat-type>
               <i0> <i1> <j0> <j1><k0><k>)
     )
     [ (isot-dir) ]
     [ (isot
          (<num><dir><i0><i1><j0><j1><k0><k1>)
               ...
          (<num><dir><i0><i1><j0><j1><k0><k1>)
     ) ]
     [ (volfac
          (<num><i0><i1> <j0> <j1> <k0><k1>)
               ...
          (<num><i0><i1><j0><j1><k0><k1>)
     ) ]
     [ (areafac
          (<num><dir><i0><i1><j0><j1><k0><k1>)
               ...
          (<num><dir><i0><i1><j0><j1><k0><k1>)
     ) ]
     [ (write-mesh "<file-name>") ]
     [ (write-grid "<file-name>") ]
     [ (write-gdef "<file-name>") ]
     [ (gdef-ext "<file-ext>") ]
     [ (non-log) ]
     [ (wrap-around) ]
)
```

DESCRIPTION

put description here???

PARAMETERS

`(coord` *<coord-type>*`)`

set type of coordinate system

<coord-type>

coordinate system type, options are:

`rect`
`cylind`

`(down` *<numx>* *<numy>* *<numz>*`)`

sets the the components of the vector pointing downward in the direction of the gravity vector. The program will internally normalize the vector to unity. Setting the components to all zero will turn off gravity in the model. The vector is always with respect to a rectangular coordinate system (X, Y, Z). For a rectangular mesh, the coordinate system coincides with the rectangular coordinate system (x, y, z) of the mesh. If the mesh is cylindrical, the vector is with respect to a coordinate system (X, Y, Z) where X is the axis defined by $\theta = 0, z = 0$; Y is the axis defined by $\theta = 90°, z = 0$, and the axis Z is defined by $r = 0$.

<numx> *<numy>* *<numz>*)

x, y, and z components of downward vector

`(dx` *<numx-0>* *<numx-1>* ... `)`
`(dy` *<numy-0>* *<numy-1>* ... `)`
`(dz` *<numz-0>* *<numz-1>* ... `)`

sets mesh divisions

<numx-0> *<numx-1>* ...
<numy-0> *<numy-1>* ...
<numz-0> *<numz-1>* ...

the mesh subdivisions in the x, y, and z coordinate directions. Numbers that are repeated can be abbreviated; for example, `3*5.0` would stand for three repeats of the numeral `5`, that is, `5.0 5.0 5.0`.

```
(mat
    (<el-name-prefix> <mat-type>
        <i0><i1><j0> <j1><k0> <k1>)
        ...
    (<el-name-prefix> <mat-type>
        <i0> <i1> <j0> <j1> <k0><k1>)
)
```

sets material property name of each element

The element names will be of the form *<el-name-prefix>*`#`*<i>*`:`*<j>*`:`*<k>* where *<i>*,*<j>*,*<k>* denote the i, j, and k indices. The material type *<mat-type>* is defined in `(rocktab ...)`. The symbols `nx`, `ny`, and `nz` can be used anywhere in place of a number where an index is required. The model interprets these to mean the number of subdivisions in the x, y, and z directions, respectively.

[`(isot-dir)`]

this parameter is optional. Affects the choice of the permeability (or hydraulic conductivity) parameter. See the documentation of the permeability parameters `K0`, `K1`, and `K2` below. This parameter should not be present for models where isotropic permeability is desired.

```
[(isot
    (<num> <dir> <i0><i1><j0><j1><k0><k1>)
        ...
    (<num> <dir> <i0> <i1><j0><j1><k0><k1>)
```

```
)]
```

sets isot = 0,1,2 in x,y,z directions respectively; default is isot = 0 for all elements. The parameter **isot** selects which of the permeability (or hydraulic conductivity) values K0, K1, K2 set in **rocktab** are used for the particular element.

```
[ (volfac
    (<num> <i0> <i1> <j0> <j1> <k0> <k1>)
      ...
    (<num> <i0> <i1> <j0> <j1> <k0> <k1>)
) ]
```

sets volume modifying factor. Multiplies the volume of the specified elements by *<num>*

```
[ (areafac
    (<num> <dir> <i0> <i1> <j0> <j1> <k0> <k1>)
      ...
    (<num> <dir> <i0> <i1> <j0> <j1> <k0> <k1>)
) ]
```

sets area modifying factor

<dir>

valid options: x, y, or z;
dir = x will mult. the area between block i,j,k and i+1,j,k
dir = y will mult. the area between block i,j,k and i,j+1,k
dir = z will mult. the area between block i,j,k and i,j,k+1

```
[ (write-mesh "<file-name>") ]
```

write out $con and $el

```
[ (write-grid "<file-name>") ]
```

write out $freegrid; if two files are same, will write to same file

```
[ (write-gdef "<file-name>") ]
```

write out minimal geometry information about the grid to file; the format of the grid definition file is

```
$gdef
$type <word>       mesh type specified in coord
$nx   <real>       no. of subdivisions in x direction
$ny   <real>       no. of subdivisions in y direction
$nz   <real>       no. of subdivisions in z direction
$order <word>      ordering of elements, e.g. xyz, yzx, or zxy
$dx                subdivisions in first coordinate
    <real>
    ...
    <real>
$dy                subdivisions in second coordinate
    <real>
```

...
<real>
`$dz` *subdivisions in third coordinate*
<real>
...
<real>

Line breaks are treated as significant in this format.

[`(gdef-ext "`*<file-ext>*`")`]

write out minimal geometry information about the grid to file with file extension

[`(non-log)`]

by default, flow areas in the radial direction for cylindrical coordinates are calculated using the logarithmic formula

$$A = \frac{1}{2}(\Delta r_{i-1} + \Delta r_i)\Delta\theta\Delta z / \ln(r_i/r_{i-1}).$$

If the `non-log` flag is present, flow areas are calculated as,

$$A = (r_i - \Delta r_i/2)\Delta\theta\Delta z.$$

[`(wrap-around)`]

if *<coord-type>* is set to `cylind`, and the angles in the y (angular) direction sum to 360°, the elements at $j = 1$ are adjacent to corresponding elements at $j = ny$ where ny is the number of subdivisions in the y direction. When this option is present the model will make connections between these elements. This option is valid only for *<coord-type>* set to `cylind`. Default is no wrap-around. (Note that the `wrap-around` option will increase the band width of the matrix when using the direct solution option or the `comb` option of the preconditioned conjugate gradient method, and, therefore, increase cpu time.)

NOTE: indices start from 1 (internally, the program converts them to start at 0)

coords of blocks start from zero and are incremented by corresponding values in `(dx ...)` `(dy ...)` and `(dz ...)`

If type is `cylind`, then the first coord x is radial distance, the second y is angle, and third z is longitudinal to central axis;

The units of the angles in the `cylind` option are in degrees.

NAME

mesh-file – specify name of mesh file

SYNOPSIS

`(mesh-file "<file>")`

DESCRIPTION

The user can set up a grid either by using the `(genmsh ...)` or by generating a mesh file outside of the program and then by reading the mesh file into the NUFT model. The `genmsh` option can only produce grids that are either rectangular or cylindrical. The advantage of a mesh file is that the user can write a program to generate the user's own grid taking full advantage of the generality of the integrated finite difference method. The format of the mesh file is described in Appendix A.

PARAMETERS

<*file*> name of mesh file

NAME

grav-factor – set gravity modification factor (optional), including gravity orientation (optional)

SYNOPSIS

`(grav-factor` *<factor>*`)`

DESCRIPTION

The user can optionally mutiply the gravity vector in the model by this factor. (If the `mesh-file` option is used, this factor is multiplied in addition to the `beta` factor read from the mesh file.) By setting the factor to zero, gravity may be turned off to see what its effect is. By setting this factor to the cosine of the angle of inclination relative to the vertical downward direction assumed when creating the mesh file, one can change the orientation of the model without rereading the mesh file. When using the `genmsh` option, one can do the same thing by setting the input parameter `down`.
The default value is unity.

PARAMETERS

<factor> real number multiplying gravitational acceleration vector (default: 1.0)

NAME

time, tstop, dt, dtmax, stepmax – set automatic time stepping parameters

SYNOPSIS

(`time` <*t-real*>)
(`tstop` <*t-real*>)
(`dt` <*t-real*>)
(`dtmax` <*t-real*>)
(`stepmax` <*integer*>)

DESCRIPTION

These input parameters are related to the automatic determination of the time step size. Except for `stepmax` their values are <*t-real*>, i.e. *time-real*. The time stepping algorithm that is used is described in Appendix B.4. These parameters are required. Subsection describes optional parameters related to the automatic time stepping and Newton Raphson iteration.

PARAMETERS

`time`	initial simulation time
`tstop`	simulation stopping time
`dt`	initial time step
`dtmax`	maximum time step allowed
`stepmax`	maximum number of time steps, if exceeded, the run will stop

NOTES

The initial simulation time step overrides the time read in from a restart file if the `restart` command is present. If the `time` command is not present, then the time read in from the restart file will be used.

NAME

tolerdt, reltolerdt, dtmin – set automatic time stepping parameters (optional)

SYNOPSIS

`(tolerdt` <*real*> <*real*> ...`)`
`(reltolerdt` <*real*> <*real*> ...`)`
`(dtmin` <*t-real*>`)`

DESCRIPTION

These input parameters are related to the automatic determination of the time step size at each time step as generated by the time stepping algorithm. The algorithm seeks to control maximum changes between the components of the solution vector from the current time step to the next. The algorithm that is used is described in Appendix B.4.

PARAMETERS

`tolerdt`

maximum tolerance for change in components of solution vector from one time step to the next. The data parameters are of the form of a list of non-negative <*real*> values, one for each type of quantity in the solution vector. For example, for a model with saturation and pressure in Pascals as the primary variables, in that order, would have the form

```
(tolerdt 0.5 1.e5)
```

See the specific model documentation for specific details and the default values.

`reltolerdt`

maximum relative tolerance for change in components of solution vector from one time step to the next. The data parameters are of the form of a list of non-negative <*real*> values, one for each type of quantity in the solution vector. For example, for a model with saturation and pressure as the primary variables, in that order, the following

```
(reltolerdt 0.1 0.2)
```

would seek to adjust the time step such that the saturation does not change more than 10% and the pressure more than 20% relative to the previous time step values. The model will use the larger of the two time steps calculated from the `tolerdt` and the `reltolerdt` values. See the specific model documentation for specific details and default values.

`dtmin`

minimum time step (default 0.0)

NAME

tolerconv, reltolerconv, itermax, iterbreak, cutbackmax – set parameters controlling Newton-Raphson iteration

SYNOPSIS

```
(tolerconv <real> <real> ...)
(reltolerconv <real> <real> ...)
(itermax <integer>)
(iterbreak <integer>)
(cutbackmax <integer>)
```

DESCRIPTION

Parameters controlling Newton-Raphson iteration (see subsection B.2 for description).

PARAMETERS

`tolerconv`

maximum tolerance for change in components of solution vector from one Newton-Raphson (NR) iteration to the next during a time step for NR convergence criteria to be satisfied. The data parameters are of the form of a list of non-negative <*real*> values, one for each type of quantity in the solution vector. For example, for a model with saturation and pressure in Pascals as the primary variables, in that order, would have the form

```
(tolerconv 0.5 1.e5)
```

See the specific model documentation for specific details and the default values. Numbers that are repeated can be abbreviated; for example, `3*0.1` would stand for three repeats of the numeral 0.1, that is, `0.1 0.1 0.1`.

`reltolerconv`

maximum relative tolerance for change in components of solution vector from one NR iteration to the next for NR convergence. The data parameters are of the form of a list of non-negative <*real*> values, one for each type of quantity in the solution vector. For example, for a model with saturation and pressure as the primary variables, in that order, the following

```
(reltolerconv 0.1 0.2)
```

would specify that NR convergence criteria is met if the saturation does not change more than 10% and if the pressure changes no more than 20% relative to the previous NR iteration values. The convergence criteria of the model is satisfied if one or both of the `tolerconv` and `reltolerconv` criteria are satisfied. See the specific model documentation for specific details and default values. Numbers that are repeated can be abbreviated; for example, `3*0.1` would stand for three repeats of the numeral 0.1, that is, `0.1 0.1 0.1`.

`itermax`

maximum NR iterations. If exceeded (i.e. if NR convergence has not been reached and the NR iterations is greater than this number) the time step size is cutback by one-half and the time step is started over again. (default value: 8)

`iterbreak`

go on to next time step if this many NR iterations has been reached regardless of whether the NR convergence criteria are met. (default value: 1000000000)

`cutbackmax`

maximum number of times in a given time step that the time step has to be started over again due to lack of NR convergence. If exceeded, the program will print and error message and then stop. (default value: 100)

NAME

linear-solver, pcg-parameters –

SYNOPSIS

```
(linear-solver <word>)
[ (pcg-parameters (precond <word>) (north <integer>)
                  (toler <real>) (itermax <integer>)
                  [ (direct <integer> <integer> ...) ]
  ) ]
[ (ilu-degree <integer>) ]
```

DESCRIPTION

These parameters are for the solution of linear system of equations that needs to be solved in each iteration of the Newton-Raphson method. Example:

```
(pcg-parameters  (precond comb) (north 10) (toler 1.e-4)
                 (itermax 30)(direct 1 0))
```

PARAMETERS

`linear-solver`

sets the method used to solve the system of linear equations in the Newton-Raphson method. Valid values are

`lublkbnd`	standard ordered, constant band width block-banded gaussian elimination (default).
`vband`	variable band width elimination with reverse cuthill-mckee bandwidth minimization.
`d4vband`	d4 ordered, variable bandwidth elimination.
`pcg`	orthomin preconditioned conjugate gradient method.

(default value: `blkbnd`)

`pcg-parameters`

parameters for preconditioned conjugate gradient (PCG) method

`precond`

type of preconditioning method used. Options:

`dkr`	first degree incomplete ILU.
`ilu`	incomplete ILU decomposition with variable fill-in.
`d4`	incomplete ILU with d4 ordering with variable fill-in.
`comb`	combinative method.
`bgs`	block gauss-seidel method.
`none`	no preconditioning.

The recommended option for problems with linear or nearly linear equations (eg. ucsat) is

```
(pcg-parameters (precond bgs)(toler 1.e-3)(itermax 500) (north 20))
```

For problems with intermediate nonlinearity (e.g. us1p, usnt) the incomplete ilu of degree 0 (no fill-in) with generalized d4 ordering is recommended,

```
(pcg-parameters (precond d4)(toler 1.e-3)(itermax 200) (north 15))
```

For problems with severe nonlinearities (e.g. usnt with phase changes or highly non-linear characteristic curves)

```
(pcg-parameters (precond d4)(toler 1.e-3)(itermax 200) (north 15))
(ilu-degree 1)
```

The degree of incomplete LU decomposition is set by ilu-degree to one. Recall that the default is zero. Another available option for problems where D4 ordering is not appropriate,

```
(pcg-parameters (precond ilu)(toler 1.e-5)(itermax 100) (north 15))
```

Note that the d4 ordering is generalized, meaning that it also works when connections between elements are non-standard with respect to the standard i,j,k lattice.

`north`
: number of orthogonalizations performed in the orthomin PCG method.

`toler`
: convergence tolerance for the PCG method. The convergence criteria is that the norm of the residual vector be less than **`toler`** times the norm of the initial residual vector. The mean square norm is used,

$$||\mathbf{r}|| = (\frac{1}{n}\sum_{i=1}^{n} r_i^2)^{1/2}$$

`itermax`
: maximum number of PCG iterations. If exceeded, the time step size will be cut back in half and the time step started over.

`direct`
: specifies which of the balance equations are solved directly in the combinative method (preconditioning option `comb`). Needed only for `comb` preconditioning method. Parameters are a list of <*integer*>'s, one for each balance equation. A non-zero value will cause the linear equations for that balance equation to be solved directly by gaussian elimination.

`ilu-degree`
: degree of fill-in for ILU decomposition. Degree zero equals no fill-in.

NAME

time-method – specify the time discretization method

SYNOPSIS

`(time-method` <*string*>`)`

DESCRIPTION

This parameter specifies whether to use either a fully-implicit or fully explicit discretization in time.

PARAMETERS

`time-method`
: Valid values are `"fully-implicit"` or `"explicit"`. (default value: model specific)

NAME

upstream-weighting – set upstream weighting

SYNOPSIS

[(`upstream-weighting` <*weight*>)]

DESCRIPTION

sets upstream weighting. To calculate the advective flux $c\mathbf{V}$ of a component between two connected elements, the model will use the weighting

$$c = ac_{up} + (1 - a)c_{down}$$

based on the weighting factor w where $a = wL_{up}/(wL_{up} + (1 - w)L_{down})$ where L refers to flow lengths, the supscript *up* refers to upstream value and *down* to downstream value. The default is $w = 1$ which is full upstream weighting.

PARAMETERS

<*weight*>	weighting factor, usually between 0 and 1 inclusive (default: 1.0)

NAME

flux-correction, flux-correct-options – set parameters controlling flux correction scheme.

SYNOPSIS

```
(flux-correction  <word>)
(flux-correct-options (method <word>) (iter <integer>))
```

DESCRIPTION

The model by default uses complete upstream weighting of mobilities to numerically calculate the advective flux between two elements (see `upstream-weighting` for changing the amount of upstream weighting). An alternative method is to use the flux correction scheme of Smolarkiewicz [1983] which is a modification of the upstream weighting. If `flux-correction` is set to `on`, then the method of Smolarkiewicz is used with three iterations. The number of iterations can be changed by setting parameters in `flux-correct-options`. An upstream modified harmonic averaging method can also be used instead of the Smolarkiewicz.

PARAMETERS

`flux-correction`
: can be set to either `on` or `off` to turn on flux-correction on or off.

`method`
: choice of flux correction method

  `smolark`
  : flux-correction using Smolarkiewicz's method

  `harmonic`
  : flux-correction using harmonic averaging

`iter`
: Number of iterations for flux-correction. Using greater than two iterations for the harmonic method can lead to spatial oscillations in the solution. Using value of one for the harmonic appears to give the same results as for the Smolarkiewicz method.

NAME

title – set run title

SYNOPSIS

`(title` <*string*>`)`

DESCRIPTION

This input parameter specifies the run title which is placed at the top of output files. Example,

```
(title "Run 3A: hydrological study")
```

NAME

output-file, output-prefix, output-ext – specifies names of the various output files

SYNOPSIS

`(output-file <string>)`
`(output-prefix <string>)`
`(output-ext <string>)`

DESCRIPTION

The model will place its output into various files. There will be at least one main output file and, possibly, several auxilliary files generated by the `output` option described in this section. There are several possibilities on how the user can specify the names of these files. One possibility is to separately name each file using the `output-file` parameter to name the main output file and the `file` parameter in `output` to name the auxilliary files. Another more convenient method is have all files having the same prefix (e.g. the run name) and have different suffixes for each file (e.g., ".out", ".pg", ".T" etc). If this latter method is used, the prefix is set using `output-prefix` and the suffix of the main output file is set using `output-ext`. The suffix for the names of the auxilliary files are set using `file-ext` described in this section on the data unit, `output`.

PARAMETERS

`output-file`
: name of main output file

`output-prefix`
: all output files including those written by the `(output ...)` data field (see ??) will have this prefix; this is usually used to specify a single run name where all output files start with this name

`output-ext`
: instead of specifying the output file one can specify the prefix of the output file

NAME

output – specifies output

SYNOPSIS

```
(output
     (field
        [ (file "<file-name>") ]
        (format <options>)
        { (range "<element range>"... ) ||
        (index-range (<i0> <i1> <j0> <j1> <k0> <k1>) ...) }
        (variables <el-var0> <el-var1> ...)
        (outtimes <t0> <t1> ...)
     )
     (flux-field
        [ (file "<file-name>") ]
        (format <options>)
        { (crange ("<element range>" "<element range>") ...) ||
        (index-crange ((<i0> <j0> <k0> <i1> <j1> <k1>) ... )) }
        (variables <con-var0> <con-var1> ...)
        (outtimes <t0> <t1> ...)
     )
     (history
        (variable <el-var>)
        (element "<element name>")
        [ (file "<file-name>") ]
     )
     (flux-history
        (variable <con-var0>)
          (comp-flux (<comp> <phase>)) }
          { (connection "<element name>" "<element name>") ||
          (index-con <i0> <j0> <k0> <i> <j> <k>) }
          [ (file "<file-name>") ]
     )
     (srcflux
          (name <src-name>)
          (comp <comp-name>)
          [ (file "<file-name>") ]
          (outtimes <t0> <t1> <t2> ...)
          [ (cumulative) ]
     )
     (bcflux
          (name <bc-name>)
          (comp <comp-name>)
          [ (file "<file-name>") ]
          (outtimes <t0> <t1> <t2> ...)
          [ (cumulative) ]
     )
     (forcetimes
          (outtimes <t0> <t1> <t2> ...)
     )

     (restart
          [ (file "<file-name>") ]
```

```
            (outtimes <t0> <t1> <t2> ...)
        )

        (extool
            (range "<element range>" ...)  ||
            (index-range (<i0> <i1> <j0> <j1> <k0> <k1>) ... )
            (variables <var0> <var1> ...)
            (outtimes <t0> <t1> <t2> ...)
            [(file "<file-name>") ]
        )
)
```

DESCRIPTION

specifies the writing of various output data to files. Any of the above options can be specified; none of them is required.

PARAMETERS

```
(field
    [ (file "<file-name>") ]
    (format <options>)
    { (range "<element range>"... ) ||
    (index-range (<i0> <i1> <j0> <j1> <k0> <k1>) ... ) }
    (variables <var0> <var1> ...)
    (outtimes <t0> <t1> <t2> ...)
)
```

outputs element data

```
(flux-field
    [ (file "<file-name>") ]
    (format <options>)
    { (crange ("<element range>" "<element range>") ...) ||
    (index-crange ((<i0> <j0> <k0> <i1> <j1> <k1>) ...)) }
    { (phase-fluxes <phase0><phase1> ...)  ||
    (comp-fluxes (<comp0> <phase0>) (<comp1> <phase1>) ...) }
    (outtimes <t0> <t1> <t2> ...))
)
```

outputs flux output data

```
(history
    (variable <var. name>)
    (element "<element name>")
    [ (file "<file-name>") ]
)
```

specifies output of element variable vs. time

```
(flux-history
    (phase-flux <phase>) ||
    (comp-flux (<comp> <phase>))
    { (connection "<element name>" "<element name>") ||
    (index-con <i0> <j0> <k0> <i1> <j1> <k1>) }
    [ (file "<file-name>") ]
)
```

specifies output of flux variable vs. time

```
(srcflux
    (name <src-name>)
```

```
    (comp <comp-name>)
    [ (file "<file-name>") ]
    (outtimes <t0> <t1> <t2> ...)
    [ (cumulative) ]
)
```

outputs the total instantaneous flux of a component, <*comp-name*>, due to a source term set in `srctab`. The name of the source term <*src-name*> is the desired one in `srctab`. The sign convention is such that flow out of the problem domain is positive and flow out of the domain is negative. If the statement [(cumulative)]is present, then the cumulative flux is outputted instead of the instantaneous flux. Note that cumulative fluxes are reset to zero at the beginning of a restart.

```
(bcflux
    (name <bc-name>)
    (comp <comp-name>)
    [ (file "<file-name>") ]
    (outtimes <t0> <t1> <t2> ...)
    [ (cumulative) ]
)
```

outputs the total instantaneous flux of a component, <*comp-name*>, flowing into a set of elements specified as boundary elements in `bctab`. The name of the boundary condition <*bc-name*> is the desired one in `bctab`. The sign convention is such that flow out of the set of elements is positive and flow into the elements is negative. If the statement [(cumulative)]is present, then the cumulative flux is outputted instead of the instantaneous flux.

```
(forcetimes
    (outtimes <t0> <t1> <t2> ...)
)
```

forces the model to hit specified times without necessarily outputting any information. Used to prevent the model from skipping over the times at which fluxes or boundary conditions change suddenly.

```
(restart
    [ (file "<file-name>") ]
    (outtimes <t0> <t1> <t2> ...)
)
```

write out a restart record at the specified times. Restart record can be read in as initial conditions using the `restart` command. Backup restarts are also written (See Subsection on reading restart information.). The `state` command must not be present when using this command.

```
(extool
    (range "<element range>" ...)  ||
    (index-range (<i0> <i1> <j0> <j1> <k0> <k1>) ... )
    (variables <var0> <var1> ...)
    (outtimes <t0> <t1> <t2> ...)
     [(file "<file-name>") ]
)
```

outputs a "time-history file" in *extool* format program.

NOTES

1. (outtimes <*t0*> <*t1*> <*t2*> ...) can be replaced by either of:

   a. (outtimes) which means all times
   b. (triggers

```
    ((wake <state-var> (range "<el-name>"...) <op> <val>)
    (cond <state-var> (range "<el-name>"...) <op> <val>))
    ...
    ((wake <state-var> (range "<el-name>" ...) <op> <val>)
    (cond <state-var> (range "<el-name>" ...) <op> <val>))
)
```

which checks to see at every time step if any of the triggers goes off ; a trigger goes off iff the condition v <*op*> <*val*> is true for the wake field where v is the value of the state variable <*state-var*> at the elements with names in the range ("<*el-name*>" ...) ; if this condition is true then the corresponding condition for the cond field is checked; if true then trigger goes off and output occurs; triggers only go off once

2. (format <*options*>) : <*options*> can be any of following list or by-list list values and element names (contsac format)

`by-set`	list values as a lisp vector, [#n v1 v2 ...vn]
`by-x`	list x coord. and value
`by-y`	list y coord. and value
`by-z`	list z coord. and value list i,j,k index `by-ijk`
`by-xtable`	format compatible with `state` using by-xtable method, user needs to comment out output header
`by-ytable`	format compatible with `state` using by-ytable method, user needs to comment out output header
`by-ztable`	format compatible with `state` using by-ztable method, user needs to comment out output header
`tabular`	multi-column format
`contour`	format readable by the `nview` program for MS-DOS

3. in all option except extool option if (file ...) or (file-ext ...) is missing then will write to output file

4. all extool options must write to a single file; only the file name of the first extool data block is read, file name of the rest are ignored if present

5. `(file "<`*file-name*`>")` can be replaced by `(file-ext "<`*file-name suffix*`>")` if the field output-prefix in the has been set; the output file name is then the suffix appended to the output-prefix

6. i,j,k indices start from 1 and go to nx,ny,or nz, resp;

`(output-flow-field (file <string>) (states Sl)(fluxes liquid))`
specifies output of flow field file: saturations and flux used by transport program
`file` file name to hold flow field information

Optional Output Information

`(print-input on)`	output input data
`(print-pcg off)`	output pcg convergence info.
`(print-mat-bal off)`	output material balance
`(print-matrix off)`	output matrix
`(print-accum off)`	output accumulation
`(print-accum-coefs off)`	output accumulation coefs.
`(print-flux off)`	output internal fluxes
`(print-flux-coefs off)`	output internal flux coefs.
`(print-sol off)`	output solution at end of each time step
`(print-sol-NR off)`	output solution between NR iterations
`(print-dsol off)`	output changes between NR iterations
`(print-NR-iter off)`	output NR iteration no. to stderr
`(print-qsrc off)`	output source terms
`(print-bc off)`	output bc terms
`(print-log off)`	turns on printing to a log file
`(print-all-terms off)`	output values of terms
`(print-eqtpar off)`	output eqtpar structure
`(print-timepar off)`	output time step data
`(print-srctab off)`	output src tables
`(print-bctab off)`	output bc tables
`(print-rocktab off)`	output rock tables
`(print-fprop off)`	output fluid properties
`(print-ifdgrid off)`	output grid data
`(print-dt-state off)`	output time step restriction
`(print-conv-state off)`	output NR convergence restriction

SYNOPSIS

```
(state
    (<state var. name>  <method>  <data>)
        ...
    (<state var. name>  <method>  <data>)
)
```

DESCRIPTION

sets initial conditions for primary variables

NOTES

This command must be left out when reading initial data from a restart file using the `restart` command.

examples of <*method*>:

```
(<state var. name> by-key ("<elem. range>"<value>)
                   ...
                          ("<elem. range>"<value>))

(<state var. name> by-set <vector>)
```

where <*vector*> = [#<*n*> <*v0*> <*v1*> ...]
and <*n*> is no. of components and (the #<*n*> field is optional); here, [and]are actual characters and do not represent delimiters of optional parameters

```
(<state var. name>  by-xtable  <table>),
(<state var. name>  by-ytable  <table>), or
(<state var. name>  by-ztable  <table>)
```

where <*table*> = (<*z0*> <*val0*> <*z1*> <*val1*> ...) is a table of values with respect to the appropriate x,y, or z coordinate

NOTES:

1. A state variable can appear more than once to overwrite values set by previous specifications; for example, values for all elements can be set by "`by-xtable`" or "`by-set`" and then "touched up" using "`by-key)`".
2. The program will terminate if a primary variable has not been set for all of the elements.

NAME

read-restart – read restart record

SYNOPSIS

```
(read-restart
     (file "<file-name>")
     [ (time <restart-time>) ]
)
```

DESCRIPTION

sets initial conditions for primary variables by reading in a restart record from a file.

PARAMETERS

"<*file-name*>" string
name of the restart file created by the option `(output ...(restart ...) ...)`.

<*restart-time*> *real*
Time used to search in the restart file; the first restart record with time ¿= <*restart-time*> will be read in. The initial time of the simulation run is set by `time` and overwrites the time read in through the restart file. If `time` is not present, the initial of the run is set to the time read in.

NOTES

Restart records are created from a previous run using the `restart` option in the `output` command.

Restart Backup

The model will periodically write out restart records so that the model can be restart in case of system failure. The model alternately writes out to two restart files named <*prefix*>.re0 and <*prefix*>.re1 where <*prefix*> is set by `output-prefix`. Each file contains a single record; previous records are overwritten. The reason for using two files instead of a single one is to prevent losing a record if the system fails during a write. The user must check the two files to see which has the record with the largest time. The model writes to a file at periodic intervals based on the wall clock time.

`(backup` <*option*>`)` *optional*

<*option*> *word* if set to `on`, then the model will periodically write backup restarts. If set to `off`, the model will not do backups. Default is `on`.

`(backup-period` <*backup-time*>`)` *optional*

<*backup-time*> t-real
wall clock time period for model to perform backup restarts. Default value is 10m (i.e., 10 minutes).

NAME

rocktab – sets rock properties

SYNOPSIS

```
(rocktab
     (<rock-type name>
          (porosity <real>)
        ...
     )
          ...
     (<rock-type name>
          ...
     )
)
```

DESCRIPTION

sets rock properties for each rock type; the rock type of each element is contained in the mesh file

PARAMETERS

<*rock-type name*>	the name of the rock type
`(porosity` <*real*>`)`	porosity

NAME

srctab – set source terms

SYNOPSIS

```
(srctab
     (compflux <comp-name>
                 (range "<elem-range>" "<elem-range>" ...)
                 (table  <flux-table>)
                 [ (enthalpy <enthalpy-table>)]
     )
     ...
     (compflux
          ...
     )

     (phaseflux <phase>
          (name <phaseflux-name>)
          (range "<elem-range>" "<elem-range>" ...)
          (table  <phase-flux-table>)
          (setcomp
               (<comp-name>
                    (table <conc-table>)
                         following only for thermal models
                    [ (enthalpy <enthalpy-table>)]
               )
               ...
               (<comp-name> internal)
               ...
               (<comp-name>
                    ...
               )
          )
     )
     ...
     (phaseflux
          ...
     )
)
```

DESCRIPTION

specifies the component flux into an element or range of elements through a table of source fluxes at specified points in time.

NOTES

Linear interpolation is used for time intervals between the table values. Positive flux is flux *into* an element; negative flux is *out* of an element.

PARAMETERS

<*comp-name*> *word*

name of the component (model specific) which will be forced out or into the element(s).

<elem-range> *pattern string*
range of elements that will have the source term flux specified by the flux table.

<flux-table> *table*
component mass flux table of the form
<t0> <q0> <t1> <q1> ...
where the time values are given by *<t0>*, *<t1>*, ..., which are of data type *t-real* and the mass flux values are *<q0>*, *<q1>*, ..., which are of data type *real* and are in units of kg/sec (except that US1P model is volumetric flux m^3/s). (See subsection regarding an important note.)

<phase> *name*
Name of the phase.

<phaseflux-name> *name*
Name of the phase flux set. Used when refering to this set within output options.

<phase-flux-table> *list of reals*
Table of mass fluxes of the phase that are of the form
<t0> <q0> <t1> <q1> ...
where the time values are given by *<t0>*, *<t1>*, ..., which are of data type *t-real* and the mass flux at these times are *<q0>*, *<q1>*, ..., which are of data type *real* and are in units of kg/sec. Linear interpolation is used for values between the time values. The last time must be greater than the end time of the run.

<conc-table> *list of reals*
Table of mole fraction concentrations of the components withing the phase stream. The table is of the form
<t0> <x0> <t1> <x1> ...
where the time values are given by *<t0>*, *<t1>*, ..., which are of data type *t-real* and the concentrations are *<x0>*, *<x1>*, ..., which are of data type *real*. An alternative is to specify the component concentrations used for the flux of applied to each respective element as that within the element itself, which is done by placing the command `(`*<comp>* `internal)` instead of `(`*<comp>* `(table` *<conc-table>*`)` ...`)`.

<enthalpy-table> *list of reals*
Table of specific molar enthalpies of the component. The table is of the form
<t0> <h0> <t1> <h1> ...
where the time values are given by *<t0>*, *<t1>*, ..., which are of data type *t-real* and the enthalpy at these times are *<h0>*, *<h1>*, ..., which are of data type *real* and have units of Joules/mole. Linear interpolation is used for values between the time values. The last time must be greater than the end time of the run.

`internal`
Calculate component and energy fluxes based on concentrations and enthalpies in the respective phase within the element instead of specifying through a table of concentrations and enthalpies.

NAME

bctab – set boundary conditions

SYNOPSIS

```
(bctab
    (<bc-name>
        (range "<elem-range>" "<elem-range>"...)
        (tables
            (<primary-var> <var-table>)
            (<primary-var> <var-table>)
            ...
        )
        [ (factor
            (<comp-name> <comp-factor-table>)
            (<comp-name> <comp-factor-table>)
            ...
        ) ]
        [ (phasefactor
            (<phase-name> <phase-factor-table>)
            (<phase-name> <phase-factor-table>)
            ...
        ) ]
    )
    ...
    (<bc-name>
      ...
    )
    ...
    (<bc-name>
        (range "<elem-range>" "<elem-range>"...)
        (clamped)
        [ (factor   (<comp-name> <factor-table>)
            (<comp-name> <factor-table>)
            ...
        ) ]
        [ (phasefactor
            (<phase-name> <phase-factor-table>)
            (<phase-name> <phase-factor-table>)
            ...
        ) ]
    )
)
```

DESCRIPTION

purpose: set boundary conditions

PARAMETERS

<bc-name> *word*

name of the boundary condition. Each boundary condition has a distinct name used for identification in diagnostic output.

"*<elem-range>*" *pattern string*

range of elements which will have the boundary condition.

<primary-var> *word*
the name of the primary variable (model specific) that is being specified by the associated table.

<var-table> *table*
table of primary variable values of form,
<t0> *<var0>* *<t1>* *<var1>* ...
where the time values, *<t0>*, *<t1>*, ..., are of data type *t-real* and the entries *<var0>*, *<var1>*, ..., are the corresponding values of the primary variable that is being specified. (See subsection regarding an important note.) Primary variable values are calculated from the table using linear interpolation.

`factor`
this data unit is *optional* and is used to modify the component mass fluxes by a factor which can depend on time and is set by a table. One use of this option is to turn off a component flux coming out or into a boundary element at certain time intervals or at all times. Note that several different components can be specified each with its own time dependent factor. Tables for all of the components must be present (the components are model-specific). Leaving the `factor` data unit out, is equivalent to specifying a modification factor of 1.0 for all time.

`phasefactor`
this data unit is *optional* and is used to modify the phase mass fluxes by a factor which can depend on time and is set by a table.

<comp-name> *word*
name of the component flux which will have its flux modified (model specific).

<phase-name> *word*
name of the phase which will have its flux modified (model specific).

<comp-factor-table> *table*
table of modification factors for component fluxes form,
<t0> *<fac0>* *<t1>* *<fac1>* ...
where the time values, *<t0>*, *<t1>*, ..., are of data type *t-real* and the entries *<fac0>*, *<fac1>*, ..., are the corresponding modification factors. (See subsection regarding an important note.) Modification factors are calculated from the table using linear interpolation.

<phase-factor-table> *table*
table of modification factors multiplying all component fluxes in the corresponding phase given by *<phase-name>*. Format is analogous to *<comp-factor-table>*

`(clamped)`
Keeps the primary variables for these elements, as set in `state`, fixed in time.

NOTES

The data: `(factor ...)` is optional; if not present, factors will be unity.
Currently, the model may in some cases choose time steps which are so large that it overshoots sharp changes in a time table. This may be a serious problem if, for example, we want to model a flux that is turned off suddenly. A solution is to use the `forcetimes`

command in `output` which forces the model to hit specified times, in this case, the times at which the boundary condition changes suddenly.
The last time value in table must be greater than or equal to the ending time of the simulation as set by `tstop` or the model will abort.

NAME

element-prefix-delimiter, element-indices-separator – set format of element names

SYNOPSIS

[(element-prefix-delimiter "<*prefix-sep*>")]

[(element-indices-separator "<*ind-sep*>")]

DESCRIPTION

change the format of element names created by **genmsh**.

NOTES

The **genmsh** command names the element according to the general format:
<*elem-prefix*>#<*i*>:<*j*>:<*k*>
where <*i*>,<*j*>,<*k*> denote the i, j, and k indices of the element and <*elem-prefix*> is set by the **mat** command inside of **genmsh**. These commands allows the user to change the separators "#" and ":" by the single character in the strings "<*prefix-sep*>"and "<*ind-sep*>", respectively.

6. NOTES

Currently, the model may in some cases choose time steps which are so large that it overshoots sharp changes in a time table. This may be a serious problem if, for example, we want to model a flux that is turned off suddenly. A solution is to use the `forcetimes` command in `output` which forces the model to hit specified times, in this case, the times at which the boundary condition changes suddenly.
The last time value in table must be greater than or equal to the ending time of the simulation as set by `tstop` or the model will abort.

7. Running Flow and Transport Sequentially

In some modules, such as the *us1* module, the flow of the fluid phases is first calculated at the beginning of each step and then the transport of contaminants is performed using the resulting the flow field. (Other modules such as *usnt* solve for flow and transport simultaneously.) When the flow and transport models are solved separately, each model has its own set of input data units in the same input data file. Data units that are common to both models can be placed in a common data unit called `(common ...)` which holds data units used by both the flow and transport models. If a data unit appears in both the `common` data unit and the particular unit belonging to the model, then the one in `common` takes precedence.

General Form for Running Flow and Transport Sequentially:

```
(common
     (title ...)
          (outputfile ...) || { (output-prefix ...)(output-ext ...) }
          (meshfile ...) || (genmsh ...)
          (time ...)
          (tstop ...)
     )

(<flow-model-name>
     (dt ...)
     (dtmax ...)
     (stepmax ...)
     (state ...)
     (rocktab ...)
     [ (output...) ]
     [ (srctab ...) ]
     [ (bctab ...) ]
)

(<transport-model-name>
     (dt ...)
     (dtmax ...)
     (stepmax ...)
     (state ...)
     (rocktab ...)
     [ (output...) ]
     [ (srctab ...) ]
     [ (bctab ...) ]
     [ (phaseprop ...) ]
     [ (compprop ...) ]
)
```

8. References

Bear, J. and Y. Bachmat. *Introduction to Modeling of Transport Phenomena in Porous Media,* Kluwer Acad. Publishers, 1991.

Behie, A. and P.K. Vinsome, Block iterative methods for fully implicit reservoir simulation, Soc. Petroleum Engineers, paper 9303, 1980.

Edwards, A.L, TRUMP: a computer program for transient and steady state temperature distributions in multidimensional systems, *National Tech. Information Service,* Springfield, VA, 1972.

Grabowski, J.W., P.K. Vinsome, R.C. Lin, A. Behie, and B. Rubin, A fully implicit general purpose finite-difference thermal model for in situ combustion and steam, Soc. Petroleum Engineers, paper 8396, 1979.

Lee, K., A. Kulshrestha, and J. Nitao, Interim Report on Verification and Benchmark Testing of the NUFT Computer Code, Lawrence Livermore National Laboratory Report UCRL-ID-113521, 1993.

Narasimhan, T.N. and P.A. Witherspoon, An integrated finite difference method for analyzing fluid flow in porous media, *Water Resour. Res., 14* 255-261, 1978.

Nitao, J.J., User's Manual for the US1P and US1C Modules of the NUFT Code, Lawrence Livermore National Laboratory Rep. ???, 1995.

Nitao, J.J., User's Manual for the UCSAT Module of the NUFT Code, Lawrence Livermore National Laboratory Rep. ???, 1995.

Nitao, J.J., User's Manual for the USNT Module of the NUFT Code, Lawrence Livermore National Laboratory Rep. ???, 1995.

Richtmyer, R.D. and K.W. Morton, *Difference Methods for Initial-Value Problems,* Interscience Pub., N.Y., 1967.

Smolarkiewicz, P.K., A fully multidimensional positive definite advection transport algorithm with small implicit diffusion, *Journal of Computational Physics, 54,* 325, 1984.

Vinsome, P.K.W., Orthomin, an iterative method for solving sparse sets of simultaneous linear equations, Soc. Petroleum Engineers, paper 5729, 1976.

A. Format of the Mesh File

The mesh file is free format and can be either of two forms

First Form

The first form uses element names to specify the connections between two elements:

`$el`
`%` *<el-name>* *<rock-type>* *<volume>*
...
`%` *<el-name>* *<rock-type>* *<volume>*
`$end`

`$con`
`%` *<el-name0>* *<el-name1>* *<isot>* *<L0>* *<L1>* *<A>* *<beta>*
...
`%` *<el-name0>* *<el-name1>* *<isot>* *<L0>* *<L1>* *<A>* *<beta>*
`$end`

Second Form
The second form uses element numbers to specify connections:

`$eleme`
`%` *<el-num>* *<el-name>* *<rock-type>* *<volume>*
...
`%` *<el-num>* *<el-name>* *<rock-type>* *<volume>*
`$end`

`$conne`
`%` *<mp0>* *<mp1>* *<isot>* *<L0>* *<L1>* *<A>* *<beta>*
...
`%` *<mp0>* *<mp1>* *<isot>* *<L0>* *<L1>* *<A>* *<beta>*
`$end`

where

<el-name>	element name (*word*)
<rock-type>	rock type of element (*word*)
<volume>	volume of element (*real*)
<el-name0>, *<el-name1>*	specifies a connection with positive flow going from element *<el-name0>* to *<el-name1>* (*word*)
<isot>	value of 0, 1, or 2 referring to which conductivity `K0, K1,` or `K2` in `(rocktab` ... is used for this connection (*integer*)
<L0>, *<L1>*	lengths of the part of the connection that lies in the first and second elements that defines the connection, respectively (*real*)
<A>	flow area of a connection (*real*)
<beta>	cosine of the angle of vertical inclination of the connection (*real*)
<el-num>	the element number starting from 0; must be consecutive (*integer*) in `$eleme`; used primarily for convenience by the user in associating connections with the elements they connect when the `$eleme` and `$conne` options are used.
<mp0>, *<mp1>*	specifies a flow connection with positive flow going from *<mp0>* to *<mp1>* (*integer*)

The advantage of the first format is that connections are easily identified by looking at the element names that they connect, especially if the names are chosen in a convenient manner. The disadvantage is that more initial setup time is needed by the model to associate the element names with the element numbers that will be used during the simulation. For greater than 4000 elements the second format is recommended.

B. Numerical Algorithms Used

B.1. Numerical Discretization

NUFT solves balance equations (see, for example, Bear and Bachmat [1991]) which are partial differential equations whose specific form depends the particular NUFT module that is being used. The balance equations are discretized in space using the integrated finite difference method (see Edwards [1972], Narasimhan and Witherspoon [1978]). Spatial discretization results in a system of ordinary differential equations of the form

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{f}(\mathbf{u}, t) \tag{1}$$

where $\mathbf{u}$ is the solution vector and $\mathbf{f}$ is a vector-valued function.

A first order implicit-in-time scheme [Richtmyer and Morton, 1967] is used to discretize the balance equations for modules with combined flow and contaminant transport. For modules where flow and transport is solved sequentially, the flow equations are solved using the implicit-in-time method, and the transport equations can be solved either implicitly or explicitly in time depending on the user's choice.

When (1) is solved using the implicit-in-time method, the resulting system of equations which must be solved at each time step are non-linear if $\mathbf{f}$ is non-linear.

$$\mathbf{F}(\mathbf{u}^{(n+1)}) \equiv \mathbf{u}^{(n+1)} - \mathbf{u}^{(n)} - \Delta t^{(n)}\, \mathbf{f}(\mathbf{u}^{(n+1)}, t^{(n+1)}) = 0 \tag{2}$$

where superscripts represent the time level.

B.2. Newton-Raphson Iteration

The non-linear system of equations, (2) is of the form

$$\mathbf{F}(\mathbf{u}) = 0 \tag{3}$$

The Newton-Raphson iterative method is used to solve the equations,

$$\mathbf{J}(\mathbf{u}_n)(\mathbf{u}_{n+1} - \mathbf{u}_n) = -\mathbf{F}(\mathbf{u}_n) \tag{4}$$

where the matrix

$$\mathbf{J}(\mathbf{u}_n) \equiv \left[\frac{\partial F_i}{\partial u_j}\right] \tag{5}$$

is the Jacobian of the function $\mathbf{F}$ evaluated at the n-th iterate, $\mathbf{u}_n$. Note that (4) can be viewed as a system of linear equations

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{6}$$

where

$$\mathbf{A} = \mathbf{J}(\mathbf{u}_n) \tag{7}$$

$$\mathbf{x} = \mathbf{u}_{n+1} - \mathbf{u}_n \tag{8}$$

$$\mathbf{b} = -\mathbf{F}(\mathbf{u}_n). \tag{9}$$

B.3. Solution of System of Linear Equations

The code has, currently, two different options for solving the system of linear equations, (4): banded gaussian elimination and preconditioned conjugate gradient methods.

B.3.1. Banded Gaussian Elimination

First, for convenience, let us rewrite (4) in the matrix form

$$\mathbf{Ax} = \mathbf{b}. \tag{10}$$

where $A = [A_{ij}]$ is an $n \times n$ matrix It is well-known that in many problems the matrix $\mathbf{A}$ is *banded,* i.e. $A_{ij} = 0$ for $|i - j| > n_b$ where nb is the matrix half-bandwidth. If $nb \ll n$ then it is advantageous to perform the gaussian elimination only on the elements within the band. This band becomes more or less filled with non-zero elements as the elimination proceeds.

Problems where $\mathbf{A}$ is banded include the case where the problem domain is discretized by a rectangular mesh and the elements are ordered either rowwise or columnwise.

B.3.2. Orthomin Preconditioned Conjugate Gradient

For the description of the orthomin algorithm and the preconditioning options see Vinsome [1976] and Behie et al. [1980].

B.4. Automatic Time Stepping

The time stepping algorithm seeks to control the maximum change in the solution vector from one time step to the next. This control can be accomplished by estimating the time step for the solution to change by a specified amount based on the solution change that ocurred in the previous two time steps. During an iteration, if the change in solution from the previous time step is too large, a re-estimation is performed and the time step started over. This method also controls to some extent the time discretization error.

We now describe the method in more detail. Let u_i be the i-th component of the solution vector at the current iteration of the Newton-Raphson method, and δu_i be equal to u_i minus the i-th component of the solution vector at the end of the previous time step. Now, let $(\delta u_i)_{max}$ be the maximum allowed change in the solution. We define the new reference time step δt by

$$\delta t = \frac{(1 + w)(\delta u_i)_{max}}{w(\delta u_i)_{max} + \delta u_i} \tag{11}$$

Here, w is a "damping" factor [Grabowski et al., 1979] that is chosen between 0 to 1 in order to prevent the time step from changing too rapidly. We used a value of 0.8. The value of δt is adjusted to stay within 0.5 to 4.0 times the time step size taken in the previous time step. If the reference time step is less than the current time step because the solution changed too much, then the current time step is replaced by 0.8 times the reference, and the Newton-Raphson is restarted. The factor 0.8 is there to allow for some margin to prevent too many restarts.

If the equations are solved using an explicit-in-time method, the time step is restricted so that the Courant-Friedrichs-Levy criteria [Richtmyer and Morton, 1967] is satisfied.